DEEP LEARNING FOR IMAGE ENHANCEMENT AND VISIBILITY IMPROVEMENT

Yola Jones

B31RA Advanced Reading

Supervisor: Alexander Belyaev

Abstract

This paper will cover the work done to produce a convolutional neural network being used to filter low-light images, using a deep learning approach.

Code Repository

https://github.com/yj48/AdvancedReading

Table of Contents

Table of Figures

Table of Tables

1. Introduction

Deep learning is a machine learning approach which is currently revolutionising a number of disciplines including image processing and computer vision. This paper will attempt to apply deep learning to image filtering, specifically low-light image enhancement. To do this, a literature review will be undertaken to analyse existing work done in this field, in an attempt to take inspiration for methods and functions which could be applied to the final network. Work will then be done to produce a fully functioning image filtering system using deep learning, which will allow the network to be trained using Supervised Learning [1], and filtered output images to be saved to a file.

Overall the network was successful, producing output images which can clearly be seen to be filtering low-light images. The network will need to be run for a greater amount of time to see the best possible results the network can output, and this has been discussed in the Results section of this report.

Project Aims

The aim of this project is to produce a neural network which can take images captured in low light conditions and output a filtered image which is exported to a file and can be viewed. The network will be trained with square greyscale 28x28 images and will filter images of the same size. The restrictions are in place because of hardware limitations which will be discussed in the Technical Discussion of this report.

2. Literature Review

Low-Level Light Enhancement

Low-level light enhancement is a common aim in the field of image processing. Images captured in low-light, or due to a low-exposure, suffer from low contrast and brightness, which both degrades the visual quality of the image and can hinder further image processing algorithms (e.g. object detection [2]). There have been many attempts to correct for low-level light noise, from regular histogram-based methods as discussed by Shen et al. in [3] to adaptive gamma correction in [2] and illumination map estimation as discussed by Guo et al. in [4]. Figure 1 shows two examples of images captured in low-light, compared to their filtered alternatives, produced by the downloadable LIME code by Guo et al. in [5].

Figure 1: Low-level light images vs filtered images

Low light images have been enhanced successfully using histogram and saturation based methods, for example Lim et al. [6] and Yoo et al. in [7]. While these methods have generated promising outputs, the running time for each process can be large. With deep learning, the network is trained (which takes significant time) and produces a filter which can be run quickly and with minimal processing, for example on a smartphone.

Deep learning has been used to filter low-light images, with successful results. Figure 2 shows the result of the method proposed by Schwartz et al. in [8].

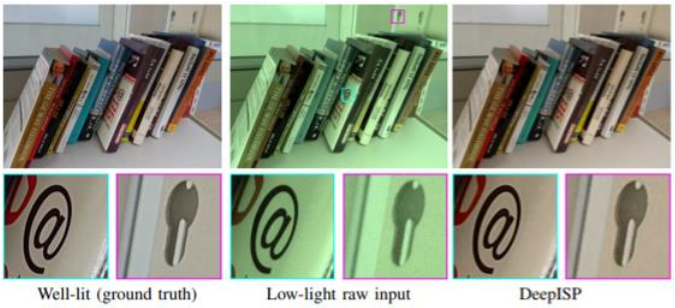

Figure 2: Low-level light filtering with deep learning

Due to the successes of using deep learning for low-level noise reduction, this paper will attempt to analyse different methods of applying deep learning to image filtering.

What is Deep Learning?

Deep learning is a branch of machine learning which attempts to replicate the human brain in machine form [9], to perform complex problems which would typically be difficult for a computer to do, such as pattern recognition. It's beginning to become more widely used in industry, with applications such as cancer detection in healthcare [10], speech recognition [11] and autonomous or partially autonomous

vehicles [12]. Deep learning has become widely used in recent years due to the increase of parallel processing capabilities through the use of GPUs [13].

Deep Learning Issues

There are a number of challenges in deep learning, as discussed by Parth Shrivastava in [14]. Hardware requirements such as the amount of data required and the amount of processor capacity required will be discussed later on in this report.

Deep learning algorithms often have issues with overfitting, when complex networks struggle to generalise the data being used to train the network with, resulting in a large overall network accuracy in training but a low accuracy when the network is tested with new unseen data [15]. A simple but processor-heavy solution to this is simply increase the batch size, which is why most papers that have been examined for this project don't mention overfitting. However, a technique called dropout regularization is commonly used and basically ignores a subset of randomly selected neurons when the network is being trained, reducing the sensitivity of the network and reducing the effects of overfitting. This has been discussed more thoroughly by Jason Brownlee in [16]. Due to time constraints this hasn't been implemented for the final network produced, but would be something to look at if the project were to continue.

In training, networks often converge further from the optimum than expected, due to the kinetic energy of the learning rate as discussed further in [17]. Essentially the learning rate as the network progresses becomes too high, causing the optimization function to jump continuously between a high and low value and never converging. Learning rate decay as discussed in [18] is a common solution, where the learning rate is reduced as the network progresses to allow the network to converge at a better optimum.

The final issue that will be discussed is the vanishing gradient problem. This is a problem which occurs in optimisation functions using gradients to minimise cost. These terms will be discussed in greater detail later on in this report. The problem is discussed more thoroughly by Muhammed Fawzy in [19], but results in neurons in the early layers of a network learning more slowly than neurons in the later layers of a network, due to back propagation. This is an issue because the early layers of a network perform fundamental operations needed throughout the rest of the network. There are solutions to this problem, with this paper focusing on the activation function to minimize the effects.

Deep Learning in Image Processing

In image processing, deep learning is often used for image classification [20] [21] but has also been used for image filtering. One of the biggest issues in applying deep learning to image processing is how to input the image data into the neural network. A simple approach would be to have each pixel being an input to the neural network. However, with a standard photograph size of 5”x7”, a typical resolution is 630x450 pixels [22]. If the image is in RGB form as opposed to grayscale, this value gets multiplied by 3. This requires a neural network with 850,500 input notes, which can rapidly increase the size of the network if multiple hidden layers are used, dramatically increasing the required computation time for the network to be trained.

To combat this issue, convolutional neural networks (CNNs) were created and will be used as the foundation of this project. These are a type of neural network which is designed specifically to be used with images, and differ slightly from traditional neural network structure. Each layer in a CNN is a 3D structure of nodes, as opposed to a 1D structure in regular neural networks. This can be seen more clearly in Figure 3.

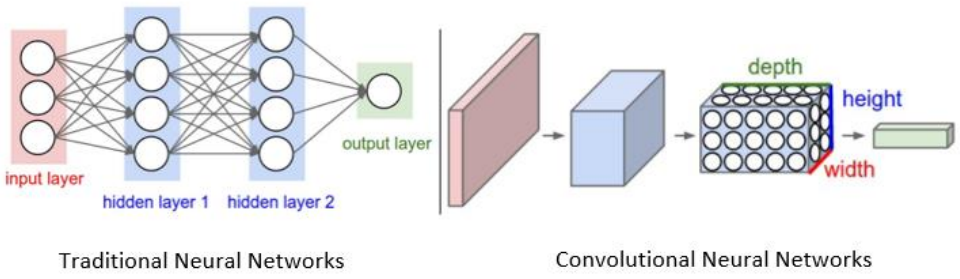

Figure 3: Traditional vs Convolutional Neural Networks [17]

Another key property of CNNs is that they are not fully connected [23] [24], where every node in a layer is connected to every node in the previous layer. There are three main elements to a CNN:

- Convolutional layer
- Pooling layer
- Fully connected layer

These will be discussed in the following sections.

Convolutional Layer

The convolutional layer is a layer (or series of layers) that consists of a sequence of filters. The nature of these filters is trained by the neural network. The image gets convolved by each one of these filters, with the output being stored in an n by n-dimensional "slice" of the next layer, the blue cuboid shown in Figure 3. An example of this can be seen in Figure 4, with the blue matrix being the input image, and the green layer being the output. It's the convolutional layers which process the image, resulting in a filtered output from the network.

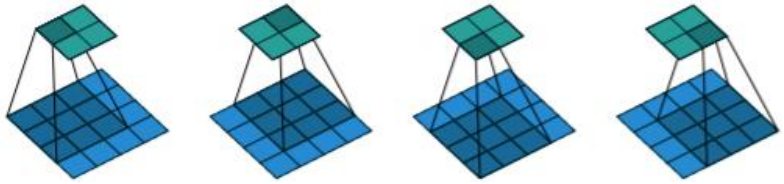

Figure 4: Convolutional Layer Example [25]

Pooling Layer

The pooling layer is simply there to reduce the dimensionality of the previous layer so it is a more appropriate size for the next layer of the network. Typically this is done with max-pooling, which takes the maximum value of the window the filter is looking at, convolved across the image [26] [27]. A CNN can have any number of convolutional and pooling layers, in any order, with the only limitations being computation power and time, and the risk of overfitting [28].

Fully-Connected Layer

The fully connected layer is a regular neural network and is typically used as the final step in a convolutional neural network being used for image classification, where the desired output is an *m* element array (with m being the number of categories of images) containing probabilities of the image being of a particular category.

Deep Learning in Image Filtering

In image filtering applications such as this, both the input and output of the CNN should be an image. Because of this, and for reasons mentioned previously when discussing why traditional neural networks are unsuitable for image processing applications, the fully-connected layer is not needed for networks

which are being trained for image filtering. This can be seen more clearly in Figure 5, where an example CNN structure for image filtering is shown.

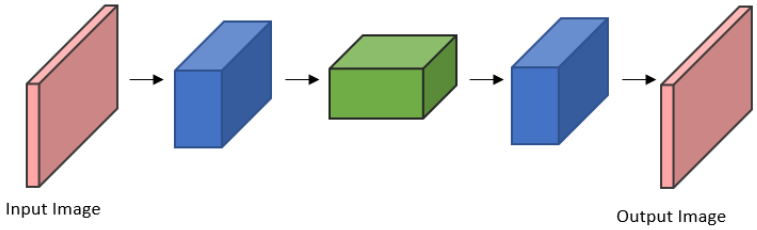

Figure 5: Image Filtering CNN

As shown in Figure 5, there is no need for a fully connected layer, since the output is a 2D or 3D image. Because of this, one will not be included in the final network structure implemented in this project.

The pooling layer may also not be needed. Since pooling is designed to reduce the dimensionality of a layer, if the desired output image is greater than or equal to the size of the input image, dimensionality reduction may not be required.

To gain a better understanding of the ideal network structure and relevant parameters, this report will look at work already done in the field of using deep learning for image filtering, in order to gain some insight into potential ways the final network could be constructed. Seven properties will be investigated:

1. The network structure, in terms of number and size of layers
2. Whether a pooling layer is used
3. The cost function
4. The activation function
5. The evaluation function
6. The number of epochs the network is being trained for
7. The batch size

Points 3 to 7 are discussed in greater detail below.

Cost Function (AKA loss function or error function)

The cost function in a network is used give feedback to the network about how badly it performed. It is this parameter which the network attempts to minimize [29] and which results in the "learning" part of deep learning. In order to train the network to minimise the result of the cost function, an optimizer must be used, with the Adam Optimizer (Adaptive Moment Estimation Optimizer) being a popular choice

[30]. Figure 6 compares the Adam Optimizer with other stochastic optimizers and can be seen to have the best results in terms of minimizing cost over time.

Ruder [31] compares this optimiser with seven other gradient descent optimisers and finds that it outperforms the others due to its inclusion of momentum and bias-correction which causes an increase in learning rate compared to the other seven methods. As well as this, because of the momentum term the optimizer also includes an element of learning rate decay, allowing the network to converge at a better optimum as discussed previously. Because of this, the Adam optimizer will be used in the final network.

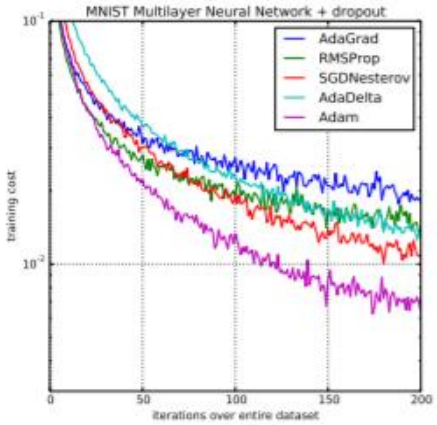

Figure 6: Adam Optimizer [22]

The Adam Optimizer is a stochastic gradient descent optimizer, which requires the cost function to be differentiable, as with all gradient descent optimisers. A differentiable cost function which is widely used is the Mean Square Error, described in more detail by Zheng in [32], and computes the average of the squared error pixel-by-pixel between the output image and the desired output image. Due to its simplicity, it is widely used in deep learning as a cost function, however, does not take any statistical features about the image into account, so isn't perfect. Due to its computational simplicity, however, when combined with an appropriate optimization function such as the Adam Optimizer, it can produce better results (according to the evaluation functions discussed below) more quickly than some of its more complex alternatives.

Zhao et al. [33] compared five different cost (or loss) functions in terms of perceptual quality to the human eye when being applied to restoration.

Activation Function

Once an image has been completely convolved with a filter from the following layer, the output is put through an activation function. The activation function is in place to introduce non-linear properties to the network [34] which allows the network to solve more complex non-linear problems than its linear alternative.

An activation function commonly used with convolutional neural networks is the Rectified Linear Unit (ReLU) [35]. It is defined as the following: $f(x) = \max(x, 0)$. Figure 7 compares the ReLU activation function to another popular activation function, the sigmoid.

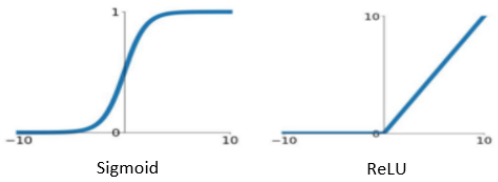

Figure 7: Sigmoid vs ReLU Activation Function [36]

As can be seen, the sigmoid activation function is prone to saturation, when the input is large the output approaches 1. With the ReLU activation function, when the input is large the output is also large. This solves the vanishing gradient phenomenon found in deep learning [37] and as discussed previously. Krizhevsky et al. [38] also found that this activation function converges much quicker than the sigmoid or tanh activation functions, making it a popular choice of function in image classification problems. However, since this activation function only inhibits values which are less than 0, it is prone to exploding gradients as discussed by Cai et al. in [39], who propose an alternative activation function called the Bilateral Rectified Linear Unit (or BReLU), and is discussed later on in this report.

Evaluation Function

The evaluation function is different to the cost function in that it's not used by the network, it's simply a parameter so we can see how well the network is doing over time.

Two common evaluation functions in image filtering applications are the Peak Signal to Noise Ratio (PSNR) and Structural Similarity Index (SSIM). Ding et al. [40] describes both of these in further detail, but in summary:

- Peak Signal to Noise Ratio (PSNR) is easy to compute and measures the cumulative squared error between the desired output image and the actual output image in decibels [41]. While it's a good cheap-and-cheerful evaluation function, it acts purely pixel-by-pixel and ignores the characteristics of the image
- Structural Similarity Index (SSIM) is a slightly more computationally complex function which again compares the desired output image and the actual output image but focuses on the structural information of the image

Another less-used evaluation function is the Naturalness Image Quality Evaluator (NIQE). This is an evaluation metric proposed by Zhang et al. in [42] and focuses on the perceived quality of an image, or how good an image looks to the human eye. Although not used that frequently it is perfectly suited to applications where the appearance of the input image will change significantly when being converted to the output image, as is the case for low-level light enhancement. A paper which uses this evaluation function as a measure of the image quality in low-level light applications will be discussed later on in this report.

Epochs and Batch Size

An epoch is a measure of how long the network will be running for. One epoch equates to the entire dataset being used to train the neural network being passed through the network a single time [43]. The batch size is how many input/output pairs the network is being presented with each epoch.

Analysis of Existing Methods

To investigate the above properties, nine papers were examined. These have been discussed in the following sections.

Haze Reduction

Cai et al. [39] propose a system for removing haze in affected images. In hazy images, the contrast between pixels is lost. The proposed system was an attempt to improve on traditional histogram- and saturation-based methods.

Their system takes an image as an input, producing a medium transmission map (a property specific to haze reduction and discussed in greater detail in [44]) which is combined with the input image using a pixel-wise operation to produce the filtered image.

Their system proposes a Bilateral Rectified Linear Unit (or BReLU) as an alternative to the commonly used ReLU activation function, discussed above. BReLU is proposed to improve image restoration accuracy compared to the ReLU, and the differences can be seen in Figure 8. Since the BReLU inhibits data both when it is less than 0 and when it is greater than t_{max}, it removes the possibility of exploding gradients which can appear while using the ReLU as discussed above.

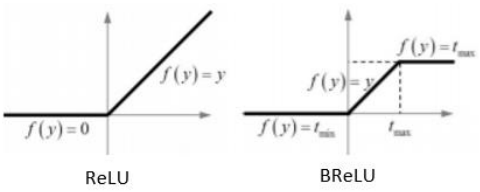

Figure 8: ReLU vs BReLU [39]

The network structure consists of three convolutional layers, one max pooling layer, a maxout unit discussed in [45], and a BReLU activation function. They use a Mean Squared Error cost function combined with a stochastic gradient descent optimisation function to train their network. They use mean squared error, structural similarity index and peak signal to noise ratio as their evaluation functions.

In order to create their training data, they select a number of noise-free images from the Internet and randomly sample them to produce the noise-free images the cost function will be compared to the output of the neural network. These samples are of pixel size 16 x 16, and then put through a filter aimed a reproducing the effects of haze.

Super Resolution

Kim et al. [46] proposed a system designed to generate high-resolution images from low-resolution images. They have a network structure of 20 layers, with each layer except the first and last being of size 3 x 3 x 64, and the final layer being a reconstruction layer used to map the previous layer to a 1-dimensional output layer the same size as the input layer. Since each hidden layer is the same size, this removes the need for max-pooling, as discussed previously. The output of their network is a residual image, which is added to the input image to produce a final filtered image. This is the same approach as used by Cai et al. in the previous example.

A Rectified Linear Unit (ReLU) activation is used, with these layers placed after each convolutional layer, and a mini-batch gradient descent optimization function is used to evaluate the cost function. The cost function is defined as the Euclidian distance between the filtered image (the sum of the input and output of the network) and the ideal noise-free image. They use Peak Signal to Noise Ratio (PSNR) and Structural Similarity Index (SSIM) as evaluation methods.

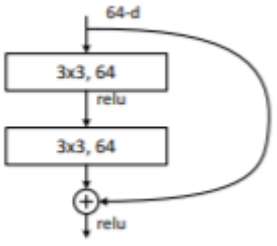

Figure 9: Residual CNN Output [39]

Both Kim et al. and Cai et al. use their networks to compute a residual image as opposed to a filtered image. This residual image is then summed with the input image to produce the filtered image. He et al. [47] put forward this idea, testing a network trained to produce a network which does exactly this, as can be seen in more clearly in Figure 9. The network generated promising results when applied to image classification.

One added advantage of this type of training is it reduces a 3-dimensional RGB image into a 1-dimensional greyscale image, which is then reconstructed into an RGB image. This reduces the size of the network significantly, by a factor of 3 for a network which takes the RGB input and immediately reduces it to greyscale. However, as mentioned by Kim et al. [46], this process works best when the input and output image is largely similar. In low-level light applications, however, the input images and output images are often largely different. Also, as previously discussed the input image to the network created for this project will be a grayscale image to reduce complexity, removing one of the advantages for generating a residual image as an output. Because of this, the output of this network will be a filtered image and not a residual image. An application which directly reduces noise due to low-level light is discussed below.

Low-Light Image Enhancement

Shen at al. [3] proposes a system for reducing noise as the result of low-level light, improving the clarity of images and allowing them to be processed further for different applications. The input to the network is an RGB noisy image, with the output being an RGB filtered image of the same size. The network consists of 10 convolutional layers and no pooling layer. They use an Adam optimizer with the angular error cost function, defined as:

$$\varepsilon = \arccos\left(\frac{\langle Y, \hat{Y}\rangle}{\|Y\|.\left\|\hat{Y}\right\|}\right)$$

Where ε is the error function or cost function, Y is the expected output and $\hat{Y}$ is the output of the network. The structural similarity index (SSIM) and naturalness image quality evaluator (NIQE) have been used as evaluation functions.

The training dataset consisted of 10,000 pairs of input/output images and was produced in the same way as Cai et al. in the DehazeNet network, with each of the 1000 images taken from the internet sampled and filtered to produce 10 input/output pairs.

Summary

For the final six papers, the key information discussed in previous sections has been extracted. The results of this can be seen in Table 1. F. in this table stands for function. This has been combined with the key information as discussed in the previous sections.

Ref	Application	Structure	Pooling?	Activation F.	Cost F.	Evaluation F.
[48]	Joint image filtering	3 x 3 layer convolutional networks	No	ReLU	Sum of squared losses	Root mean squared error
[49]	Removing rain from a single image	Multiple convolutional, batch normalization and ReLU layers	No	ReLU	MSE	SSIM
[50]	Image Super-Resolution	Two convolutional layers and a reconstruction layer	No	ReLU	MSE	PSNR
[51]	Image denoising	Multiple convolutional, batch normalization and ReLU layers	No	ReLU	MSE	PSNR
[52]	Demosaicing	Multiple convolutional layers, each followed by a ReLU layer	No	ReLU	Normalised error	PSNR
[53]	Low-light image enhancement	A combination of convolutional and pooling layers	Yes	ReLU, tanh	Multi-scale SSIM	Mean Opinion Score
[39]	Dehazing	Three convolutional layers, max pooling layer, maxout unit	Yes	BReLU	MSE	MSE, SSIM and PSNR
[46]	Image super-resolution	20 convolutional layers, reconstruction layer	No	ReLU	Euclidian distance	PSNR, SSIM
[3]	Low-light image enhancement	10 convolutional layers	No	ReLU	Angular error	SSIM, NIQE

Table 1: Analysis of CNN Parameters for Image Filtering Applications

As can be seen, despite the reservations suggested by Cai et al. [39], the rectified linear unit is the most common activation function in these papers. Because of this, and combined with the property of the ReLU which removes the effect of the vanishing gradient problem as discussed previously, the ReLU will be used as the activation function for this project. Similarly with mean squared error (MSE), despite it not taking structural information into account, it is widely used and the simplest of all cost functions mentioned so will be used as the cost function for this project.

The structure of the networks varies widely between networks. Because of this, for this project, an attempt will be made to find a simple network which maintains the same shape throughout the network, without having to perform max pooling. This is purely because it is the easiest to implement, and will allow time to be spent on other areas of the project.

Finally, for the evaluation function, despite PSNR being a commonly used evaluation function, in this case, the evaluation functions used in the low-level-light noise reduction network proposed by Shen et al. [3] will be used, as these give a clear indication of the best results specific to low-level-light enhancement. The implementation for the NIQE is relatively unclear, so the structural similarity index will be the main focus for this project.

3. Technical Discussion

TensorFlow

In order to build and train the network, the platform TensorFlow was used. As described on their website, "TensorFlow is an open source library for numerical computation using dataflow graphs" [54]. It uses multidimensional arrays known as tensors to represent data and was developed by Google Brain [55], being specifically designed to be used in machine learning. It can run on a CPU or GPU and is a large-scale system designed to be run in a number of different environments. Further information can be found in the paper by the Google Brain team who designed the system at [56]. TensorFlow has been used in a wide range of machine learning applications, such as intelligent mobile interfaces, data classification and image filtering. Examples of such applications can be found at [57], [58], [59], and [60]. TensorFlow supports multiple languages, but Python has been used for this project as it seems to be the most widely used and most supported by TensorFlow documentation.

Hardware Limitations

One of the key limitations of this project which was immediately expected was with hardware. Dong et al. in [50] used a GTX 770 GPU, with training taking 3 days. The only computer I had access to did not have a GPU, just a standard dual-core 2.8GHz CPU.

An article testing the performance of a CPU vs a GPU in TensorFlow [61] found that a good graphics card (GeForce 940MX) processed almost 3 times more samples in a second compared to a standard desktop

CPU (i7-7500U). By this logic, the network trained in [50] could take up to 9 days to train on the desktop I was using. Because this was a 12-week project, and in this time the topic of deep learning needed to be researched as well as building up a network, this was expected to be a real challenge. As well as this, the CPU I had access to wasn't a private computer, 21 other people had access to, and would be using, the same computer, so I couldn't guarantee that all processor power was being allocated to running the network.

As well as this, and as discussed in [14], the amount of input data required poses another difficulty. Cai et al. in [39] used an input dataset size of 10,000 images in order to generate successful results. In the network I was using, each image took around 10 seconds to load into the program, which would take almost 28 hours to load.

This meant that the size of input and output images had to be restricted, so results could be generated which proved the network works. Images were limited to 28x28 grayscale images. On the CPU being used, a network with a batch size of 586 images and 100 epochs, the network took just over 5 days and 3 hours to train.

Implementation

Network Structure

The initial implementation is based off a Convolutional Neural Networks Tutorial in TensorFlow [62]. This is a tutorial which takes images and labels from the MNIST dataset [63] [64] provided by TensorFlow and trains the network to classify handwritten digits (0-9) into numbers. This network has multiple alternating convolutional and pooling layers, which get fed into a fully connected layer. This fully connected layer outputs to a binary 10-element layer, with all values in the array 0, except the value with the highest probability, telling the user that the handwritten digit has been recognised as a digit of that value.

The MNIST dataset is imported directly from TensorFlow, with each of the images being grayscale 28x28 pixel images.

Since the implemented network was being used for image classification as opposed to image filtering, the network needed to be modified to allow images to be input, and images of the same size to be

output. To do this, inspiration was taken from [65], which is a convolutional neural network being used for Super Resolution. This example uses Caffe (a machine learning alternative to TensorFlow [66]) and is based on the work done by Dong et al. [50]. The network structure has been taken from this example, and consists of the following:

- Three convolutional layers, all designed to perform different functions
    1. Patch extraction: takes the input image and breaks it down into a sequence of feature maps using the different convolutional layers of the network
    2. Non-linear mapping: takes the output from the previous layer and non-linearly maps it to another high-dimensional vector
    3. Reconstruction: takes the output from the non-linear mapping layer and converts it into the final output image. This is compared to the expected output using the cost function
- After the first and second convolutional layers, activation using the Rectified Linear Unit (ReLU) activation function is performed.

The article cites and gives an example of an updated version of the work done by Dong et al. in 2014 from the same authors in 2016 [67]. Despite the results [68] showing this updated network slightly outperforms the original network, the original network has been used for this project. The network is more complex, with more layers that would be slightly more difficult to implement. As well as this, the network is designed for super-resolution, not low-level light enhancement so there is no guarantee that the same results (in terms of the new network outperforming the old one) would apply to this case.

From this, the implementation provided in [62] was adapted to this new network structure. Some further modification was done to change the cost function and evaluation function for image comparison (as opposed to label comparison). As well as this, the mechanism which fed input images and labels which could be compared to the network output needed to be changed.

Training Data Creation

To create the training data, a similar approach to the approaches used in [39] and [3] was used, where multiple images are taken from the internet and then sampled at random points to produce the desired output data. Whereas these papers filter the images to produce noisy versions of the originals, this project involved taking both the filtered and noisy images from the dataset provided by Schwartz et al.

[53], sampling both of them at the same random points to create 28x28 blocks, and then saving the image as a png in a format the network can process.

The format of the input/expected output image is taken from Isola et al. in [69] [70], where the input and expected output image is saved as one file of dimensions 2n by n (where n is the length/height of the image in pixels) with the unfiltered and filtered images next to each other. This can be seen more clearly in Figure 10. A Matlab script was created to generate these images.

Figure 10: Dataset Generation

Input/Output Data

Since the original tutorial being used to build the network took input images directly from the MNIST dataset provided by TensorFlow, the program needed to be changed to allow images to be imported from a file, processed and then converted into a tensor. This ended up causing quite a bit of difficulty, with a solution being taken from an article posted on GitHub by Alan Gray [71], where he creates his own array of images and imports them to TensorFlow. Processing the data once files had been saved to TensorFlow was taken from Isola et a. in [70] as before.

A similar issue occurred when trying to output an example of the filtered input image to a png file, so a user could see the results. Eventually, a solution was found based on the StackOverflow answers [72] and [73]. The output data is in png format of size 28 x 28 pixels.

Cost Function

To determine the network error, the Mean Square Error function was used, by combining the TensorFlow functions *tf.square()* and *tf.reduce_mean()*. This output was then put through the Adam

Optimizer discussed above using *tf.train.AdamOptimizer()* with a learning rate of 0.0001 taken from the original tutorial in [62]. The network is currently taking a batch size of 330 images and running for 50 epochs. Each epoch takes around 24 minutes to complete, resulting in a predicted training time of 20 hours.

[74] defines the mean square error as:

$$MSE = \frac{1}{MN}\sum_{n=1}^{M}\sum_{m=1}^{N}[\hat{g}(m,n) - g(m,n)]^2$$

Where $\hat{g}(x,y)$ is the expected output and $g(m,n)$ is the actual output generated by the network. When both images are equal, $\hat{g}(m,n) - g(m,n) = 0$, making the rest of the equation zero. The minimum mean squared error is 0, and we will be aiming for a mean square error of 0.

Evaluation Function

The structural similarity index (SSIM) has been used as the evaluation metric for this project and is defined by Wang et al. [75] as:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)}$$

Where:

μ_x is the average of x and μ_y is the average of y

σ_x^2 is the variance of x and σ_y^2 is the variance of y

σ_{xy} is the covariance of x and y

$c_1 = (k_1 L)^2$ and $c_2 = (k_2 L)^2$

L is the dynamic range of pixels (typically $2^{(number\ of\ pixels)} - 1$, in this case 255 for an 8-bit image)

$k_1 = 0.01$ and $k_2 = 0.03$

When both images are exactly the same, the SSIM returns a value of 1. This is documented in [75].

This is one of the only functions needed that was not already provided by TensorFlow. To rectify this, an attempt was made to write a python script that would calculate this SSIM and return a floating point value. However, despite managing to implement the function in a script separate from my main running, I could not get it to work as a callable function from within the TensorFlow session. Because of this, the implementation has been taken from a response posted on StackOverflow, which can be found here

[76]. I have not yet given up on the SSIM function, and will continue to work on it until it reaches a usable state. The function so far can be found in the Github repository linked to at the start of this paper.

Another evaluation function I wanted to use was the NIQE, the naturalness image quality evaluator, as it is designed to assess the perceptual quality of the image, or how good it looks to the human eye. Mittal et al. [77] puts forward this function and defines it as the distance between the natural scene statistic (NSS) and the multivariate Gaussian model (MVG).

Due to the complexity of the function in terms of implementation, this has not yet been attempted but would be a goal if the project were to continue.

4. Results

The final network was trained for 100 epochs, each taking 586 batches. It ran for just over 5 days and 3 hours. After each epoch, the network was tested against a common image, with the output converted to png format and saved to a file. An example of how the network progressed over the first 45 epochs can be seen in Figure 11, with the input image, expected output image, network output at epoch 1 and network output at epoch 99 shown at the bottom of the figure.

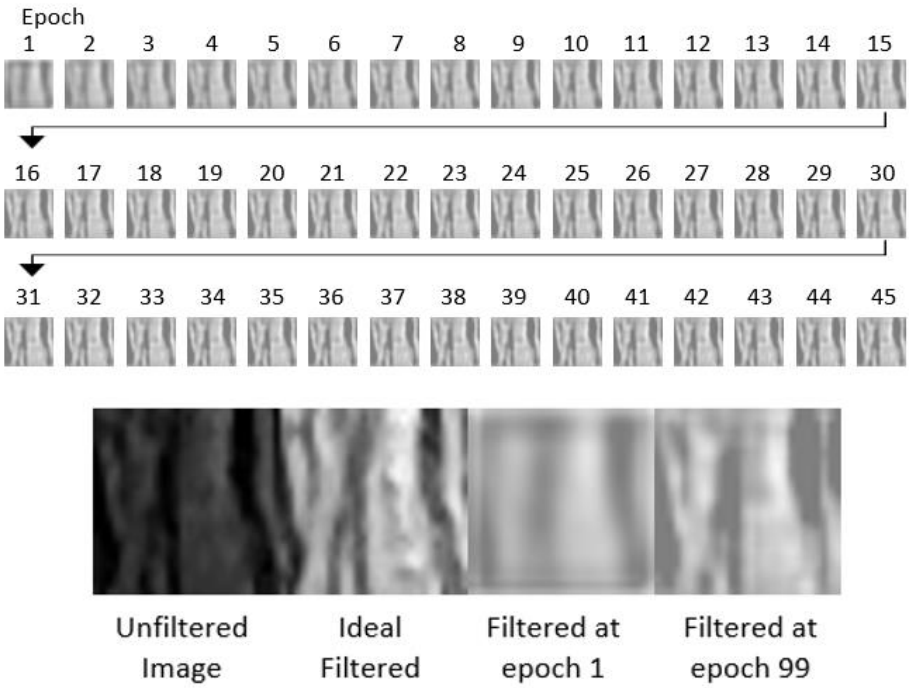

Figure 11: Final Network Image Results

As can be seen, epoch 1 brightened the unfiltered image but was blurry with lower contrast in the image than expected. Over time, the network tries to improve the contrast and sharpen the image. This can be shown more clearly in Figure 12, when the histograms of the original input image and the filtered image at epoch 99 are shown.

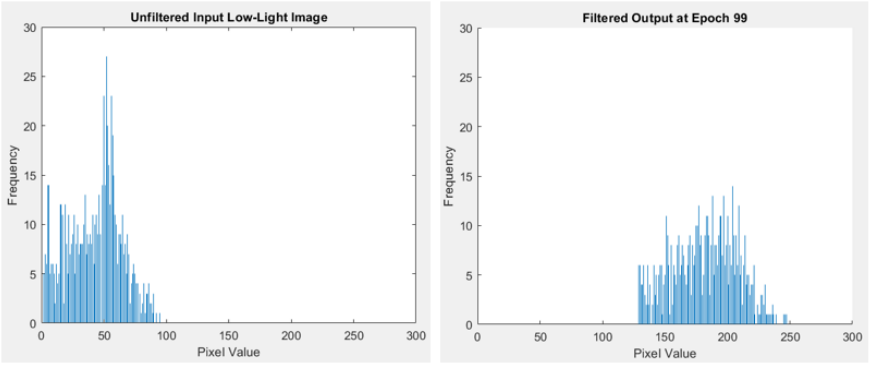

Figure 12: Histograms of Unfiltered vs Filtered Image

As can be seen, the image has been brightened considerably and the contrast improved.

Although there are clear differences between the ideal filtered image and the result at epoch 99, considering the network was only run for 5 days on a standard CPU the results are significant. The network is attempting to filter the input image and reduce the effects of low light, and the progression can clearly be seen across the epochs from the filtered image at epoch 1. The final output has been brightened and the contrast improved from the original data. The results of the mean squared error cost function and the SSIM evaluation function can be seen in Figure 13 and Table 2. The results in Figure 13 were plotted using TensorBoard, TensorFlow's in-built visualization software.

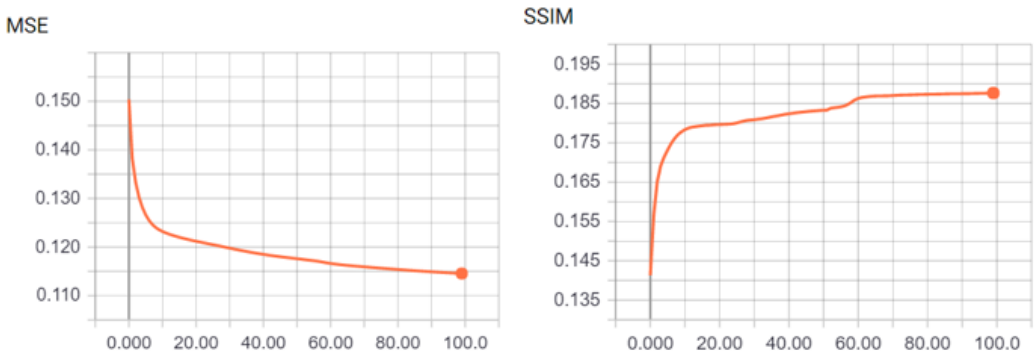

Figure 13: Final Network MSE and SSIM Plot

Epoch	MSE	SSIM	Time from start
0	0.154	0.1413	0s
20	0.1212	0.1796	1d 1h 28m 11s
40	0.1185	0.1824	2d 2h 6m 49s
60	0.1166	0.1863	3d 2h 17m 17s
80	0.1154	0.1873	4d 2h 51m 16s
99	**0.1146**	**0.1876**	**5d 3h 2m 35s**

Table 2: Final Network MSE and SSIM Values

Table 2 shows the network reached a final mean square error value of 0.1146 and structural similarity index value of 0.1876. Figure 13 shows that the difference in values between SSIM and MSE over time were decaying exponentially, and the network would likely reach a final MSE value of ~0.114 and SSIM of ~0.188. This is phenomenon appears frequently when training neural networks, and can be seen in Figure 6 with the Adam Optimizer results at the start of this paper. It is attributed to the kinetic energy of the learning rate, as discussed in [78]. A solution to this is using learning rate decay, as discussed in [18], which should allow the MSE to converge at a value closer to 0, and the SSIM to converge at a value closer to 1. Although the Adam Optimizer does include an element of learning rate decay, the decay may need increased to allow convergence of the mean squared error at a smaller value.

5. Future Work

Now that a network has been built which can take low-light images and output filtered versions of that image, work will be done to allow the network to process larger images where more significant results can be seen. This will require a lot more training time, meaning future work will likely lean towards optimisation of the network, using different network structures, cost functions and learning rates to attempt to maximise cost reduction per unit of computation time. As well this, learning rate decay will be experimented with to allow the cost and evaluation functions to converge more closely to their minimum and maximum values respectively. The network will also be tested for overfitting as previously discussed, and dropout regularization will be used to attempt to minimize the effects. Work will also be done to fix the SSIM function and implement the NIQE evaluation function as discussed above, in order to better evaluate the perceptual quality of output images.

6. Conclusion

This report has documented the work undertaken throughout the B31RA Advanced Reading course, with the topic of applying deep learning methods to image processing applications. A huge amount was learned throughout this project, specifically with regards to neural networks and the various different properties such as cost and evaluation functions, potential difficulties and pitfalls, and the wide range of applications deep learning can be used for. There were difficulties faced throughout the project which had to be dealt with, specifically with regards to hardware limitations which have been discussed, and which have caused the final network to be limited in terms of the size of the images the network can filter.

Overall the project was a success, the final network produced for this project has been proven to be able to improve the brightness and contrast of a 28x28 pixel grayscale image being taken in low light, and therefore can filter improve the perceptual quality of low-light images. Further work would need to be done to see the full effects of the network, since 28x28 pixels is much smaller than a typical image, but this would require an extensive amount of training time.

A massive amount has been gained from this project, both in terms of theoretical knowledge gained in researching the properties and applications of neural networks and how best to apply them, and practical experience in terms of programming using TensorFlow and analysing output data. The experience gained will genuinely help me through the rest of my career, and I would very much like to continue the project beyond the final submission date.

References

[1] P. Cunningham, M. Cord and S. J. Delany, “Supervised Learning,” in *Machine Learning Techniques for Multimedia Case Studies on Organization and Retrieval*, Springer, 2008, p. Chapter 2.

[2] G. Cao, L. Huang, H. Tian, X. Huang, Y. Wang and R. Zhi, “Contrast enhancement of brightness-distorted images by improved adaptive gamma correction,” *Computers & Electrical Engineergin,* vol. 66, pp. 569-582, 2018.

[3] L. Shen, Z. Yue, F. Feng, Q. Chen, S. Liu and J. Ma, “MSR-net:Low-light Image Enhancement Using Deep Convolutional Network,” ARXIV, 2017.

[4] X. Guo, Y. Li and H. Ling, “LIME: Low-Light Enhancement via Illumination Map Estimation,” *IEEE Transactions on Image Processing,* vol. 26, no. 2, pp. 982-993, 2016.

[5] X. Guo, “LIME: Low-light IMage Enhancement via Illumination Map Estimation,” Google, [Online]. Available: https://sites.google.com/view/xjguo/lime. [Accessed 17 April 2018].

[6] J. Lim, J.-H. Kim, J.-Y. Sim and C.-S. Kim, “Robust contrast enhancement of noisy low-light images: Denoising-enhancement-completion,” in *2015 IEEE International Conference on Image Processing (ICIP)*, Quebec City, 2015.

[7] Y. Yoo, J. Im and J. Paik, “Low-Light Image Enhancement Using Adaptive Digital Pixel Binning,” *Sensors - Open Access Journal,* pp. 14917-14931, 15 September 2015.

[8] E. Schwartz, R. Giryes and A. M. Bronstein, “DeepISP: Learning End-to-End Image Processing Pipeline,” 2018.

[9] J. Brownlee, “What is Deep Learning,” Machine Learning Mastery, 16 Aug 2016. [Online]. Available: https://machinelearningmastery.com/what-is-deep-learning/. [Accessed 24 March 2018].

[10] A.-R. Ali, “Deep Learning in Oncology - Applications in Fighting Cancer,” TechEmergence, 1 Sept 2017. [Online]. Available: https://www.techemergence.com/deep-learning-in-oncology/. [Accessed 24 March 2018].

[11] V. Vanhoucke, “Speech Recognition and Deep Learning,” Google, 6 Aug 2012. [Online]. Available: https://research.googleblog.com/2012/08/speech-recognition-and-deep-learning.html. [Accessed 24 March 2018].

[12] E. Ackerman, “How Drive.ai Is Mastering Autonomous Driving with Deep Learning,” IEEE, 10 March 2017. [Online]. Available: https://spectrum.ieee.org/cars-that-think/transportation/self-driving/how-driveai-is-mastering-autonomous-driving-with-deep-learning. [Accessed 24 March 2018].

[13] H. H. Eckerson, “Deep Learning - Past, Present and Future,” Eckerson Group, May 2017. [Online]. Available: https://www.kdnuggets.com/2017/05/deep-learning-big-deal.html. [Accessed 24 March 2018].

[14] P. Shrivastava, “Challenges in Deep Learning,” Hackernoon, 13 Sept 2017. [Online]. Available: https://hackernoon.com/challenges-in-deep-learning-57bbf6e73bb. [Accessed 17 April 2018].

[15] Elite Data Science, “Overfitting in Machine Learning: What It Is and How to Prevent it,” Elite Data Science, 7 Sept 2017. [Online]. Available: https://elitedatascience.com/overfitting-in-machine-learning. [Accessed 17 April 2018].

[16] J. Brownlee, “Dropout Regularization in Deep Learning Models with Keras,” Machine Learning Mastery, 20 June 2016. [Online]. Available: https://machinelearningmastery.com/dropout-regularization-deep-learning-models-keras/. [Accessed 17 April 2018].

[17] CS231n, “Convolutional Neural Netowrks (CNNs/ConvNets),” Github, [Online]. Available: http://cs231n.github.io/convolutional-networks/. [Accessed 24 March 2018].

[18] Coursera, “Learning rate decay,” Coursera, 2018. [Online]. Available: https://www.coursera.org/learn/deep-neural-network/lecture/hjgIA/learning-rate-decay. [Accessed 29 March 2018].

[19] M. Fawzy, “The Vanishing Gradient Problem,” Medium, 1 Jun 2017. [Online]. Available: https://medium.com/@anishsingh20/the-vanishing-gradient-problem-48ae7f501257. [Accessed 18 April 2018].

[20] L. P. Valem and D. C. G. Pedronette, “Selection and Combination of Unsupervised Learning Methods for Image Retrieval,” CBMI, Florence, 2017.

[21] J. Blahtua, T. Soukip and J. Martinu, “An Expert System Based on Using Artificial Neural Network and Region-Based Image Processing to Recognition Substantia Nigra and Atherosclerotic Pleques in B-Images: A Prospective Study,” *Advances in Computational Intelligence,* pp. 236-245, 18 May 2017.

[22] Walgreens Photo Help, “Image Resolution and DPI Requirements,” Walgreens, 24 July 2017. [Online]. Available: http://custhelp2.walgreens.com/articles/Information/Image-Resolution-Requirements. [Accessed 24 March 2018].

[23] IPFS, “Multilayer Perceptron,” IPFS, May 2017. [Online]. Available: https://ipfs.io/ipfs/QmXoypizjW3WknFiJnKLwHCnL72vedxjQkDDP1mXWo6uco/wiki/Multilayer_perceptron.html. [Accessed 24 March 2018].

[24] U. Karn, “An Intuitive Explanation of Convolutional Neural Networks,” 11 Aug 2016. [Online]. Available: https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/. [Accessed 24 March 2018].

[25] V. Dumoulin and F. Visin, “A guide to convolution arithmetic for deep learning,” Cornell University Library, 2016.

[26] Stanford University, “Pooling,” Stanford University, [Online]. Available: http://ufldl.stanford.edu/tutorial/supervised/Pooling/. [Accessed 24 March 2018].

[27] A. Deshpande, “A Beginner's Guide to Understanding Convolutional Neural Networks Part 2,” Github, 29 July 2016. [Online]. Available: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/. [Accessed 24 March 2018].

[28] MathWorks, “Improve Neural Network Generalization and Avoid Overfitting,” MathWorks, [Online]. Available: https://uk.mathworks.com/help/nnet/ug/improve-neural-network-generalization-and-avoid-overfitting.html. [Accessed 24 March 2018].

[29] C. McDonald, “Machine learning fundementals (I): Cost functions and gradient descent,” Towards Data Science, 27 Nov 2017. [Online]. Available: https://towardsdatascience.com/machine-learning-fundamentals-via-linear-regression-41a5d11f5220. [Accessed 24 March 2018].

[30] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” in *3rd International Conference for Learning Representations*, San Diego, 2015.

[31] S. Ruder, “An Overview of gradient descent optimization algorithms,” Cornell University Library, 2016.

[32] S. Zheng, “Methods of Evaluating Estimators,” 2016. [Online]. Available: http://people.missouristate.edu/songfengzheng/teaching/mth541/lecture%20notes/evaluation.pdf. [Accessed 24 March 2018].

[33] H. Zhao, O. Gallo, I. Frosio and J. Kautz, “Loss Functions for Image Restoration with Neural Networks,” *IEEE Transactions on Computational Imaging,* vol. 3, no. 1, pp. 47-57, 2017.

[34] A. S. Walia, “Activation functions and it's types - Which is better?,” Towards Data Science, 29 May 2017. [Online]. Available: https://towardsdatascience.com/activation-functions-and-its-types-which-is-better-a9a5310cc8f. [Accessed 24 March 2018].

[35] P. Ramachandran, B. Zoph and Q. V. Le, “Searching for Activation Functions,” Cornell University Library, 2017.

[36] A. Nikishaev, “How to debug neural networks. Manual.,” Machine Learning, 22 Aug 2017. [Online]. Available: https://medium.com/machine-learning-world/how-to-debug-neural-networks-manual-dc2a200f10f2. [Accessed 24 March 2018].

[37] M. A. Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.

[38] A. Krizhevsky, I. Sutskever and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” *NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems,* vol. 1, pp. 1097-1105, 2012.

[39] B. Cai, X. Xu, K. Jia, C. Qing and D. Tao, “DehazeNet: An End-to-End System for Single Image Haze Removal,” Cornell University Library, 2016.

[40] W. Ding, Y. Tong, Q. Zhang and D. Yang, “Image and video quality assessment using neural network and SVM,” *Tsinghua Science and Technology,* vol. 13, no. 1, pp. 112-116, 2008.

[41] PSNR, “PSNR,” MathWorks, [Online]. Available: https://uk.mathworks.com/help/vision/ref/psnr.html. [Accessed 24 March 2018].

[42] L. Zhang, L. Zhang and A. C. Bovik, “A Feature-Enriched Completely Blind Image Quality Evaluator,” *IEEE Transactions on Image Processing,* vol. 24, no. 8, pp. 2579-2591, 2015.

[43] S. Sharma, “Epoch vs Batch Size vs Iterations,” Towards Data Science, 23 Sept 2017. [Online]. Available: https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9. [Accessed 24 March 2018].

[44] F. Guo, J. Tang and X. Xiao, “Foggy Scene Rendering Based on Transmission Map Estimation,” *International Journal of Computer Games Technology,* vol. 2014, p. 13, 2014.

[45] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville and Y. Bengio, “Maxout Networks,” *Journal of Machine Learning Research,* vol. 28, no. 3, pp. 1319-1327, 2013.

[46] J. Kim, J. K. Lee and K. M. Lee, “Accurate Image Super-Resolution Using Very Deep Convolutional Networks,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[47] K. He, X. Zhang, S. Ren and J. Sun, “Deep Residual Learning for Image Recognition,” in *IEEE Conference on Computer Vision and Pattern Recognition*, 2016.

[48] Y. Li, J.-B. Huang, N. Ahuja and M.-H. Yang, “Joint Image Filtering with Deep Convolutional Networks,” ARXIV, 2017.

[49] X. Fu, J. Huang, D. Zeng and J. Paisley, “Removing Rain from Single Images via a Deep Detail Network,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

[50] C. Dong, C. C. Loy, K. He and X. Tang, “Learning a Deep Convolutional Network for Image Super-Resolution,” in *European Conference on Computer Vision*, 2014.

[51] K. Zhang, W. Zuo, Y. Chen, D. Meng and L. Zhang, “Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising,” *IEEE Transactions on Image Processing,* vol. 26, no. 7, pp. 3142-3155, 2017.

[52] M. Gharbi, G. Chaurasia, S. Paris and F. Durand, “Deep joint demosaiking and denoising,” *ACM Transactions on Graphics,* vol. 35, no. 6, 2016.

[53] E. Schwartz, R. Giryes and A. Bronstein, “DeepISP: Learning End-to-End Image Processing Pipeline,” Github, [Online]. Available: https://elischwartz.github.io/DeepISP/. [Accessed 25 March 2018].

[54] Google, “TensorFlow,” Google, 2018. [Online]. Available: https://www.tensorflow.org. [Accessed 25 March 2018].

[55] Research at Google, “Google Brain Team,” Google, [Online]. Available: https://research.google.com/teams/brain/. [Accessed 25 March 2018].

[56] Google Brain Team, “TensorFlow: A System for Large-Scale Machine Learning,” in *12th USENIX Symposium on Operating Systems Design and Implementation*, Savannah, 2016.

[57] F. Ertam and G. Aydin, “Data classification with deep learning using Tensorflow,” in *2017 International Conference on Computer Science and Engineering (UBMK)*, 2017.

[58] Google Research, “TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,” 2015.

[59] S. Mayer, H. V. Le and N. Henze, “Machine learning for intelligent mobile user interfaces using TensorFlow,” in *The 19th International Conference*, 2017.

[60] A. Kane, A. Lemionet and F. Shemaj, “Photo Style Transfer in Tensor Flow,” Stanford University.

[61] A. Lazorenko, “TensorFlow performance test: CPU VS GPU,” Medium, 27 Dec 2017. [Online]. Available: https://medium.com/@andriylazorenko/tensorflow-performance-test-cpu-vs-gpu-79fcd39170c. [Accessed 25 March 2018].

[62] Andy, “Convolutional Neural Networks Tutorial,” AdventuresInMachineLearning, 24 April 2017. [Online]. Available: http://adventuresinmachinelearning.com/convolutional-neural-networks-tutorial-tensorflow/. [Accessed 25 March 2018].

[63] Y. LeCun, C. Cortes and C. J. Burges, “The MNIST Database of handwritten digits,” [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed 25 March 2018].

[64] L. Deng, “The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web],” *IEEE Signal Processing Magazine,* vol. 29, no. 6, pp. 141-142, 2012.

[65] A. V., “An Example of a Convolutional Neural Network for Image Super-Resolution,” Intel, 28 June 2017. [Online]. Available: https://software.intel.com/en-us/articles/an-example-of-a-convolutional-neural-network-for-image-super-resolution. [Accessed 25 March 2018].

[66] Y. Jia, “Caffe,” BAIR, [Online]. Available: http://caffe.berkeleyvision.org/. [Accessed 25 March 2018].

[67] C. Dong, C. C. Loy and X. Tang, “Accelerating the Super-Resolution Convolutional Neural Network: Supplementary FIle,” in *Computer Vision - ECCV 2016*, Springer International Publishing, 2016.

[68] C. Dong, C. C. Loy and X. Tang, “Accelerating the Super-Resolution Convolutional Neural Network : Supplementary File,” Hong Kong, 2016.

[69] P. Isola, J.-Y. Zhu, T. Zhou and A. A. Efros, “Image-to-Image Translation with Conditional Adversarial Networks,” Cornell University Library, 2016.

[70] P. Isola, J.-Y. Zhu, T. Zhou and A. A. Efros, “Tensorflow port of Image-to-Image Translation with Conditional Adversarial Nets,” Github, 30 Dec 2017. [Online]. Available: https://github.com/affinelayer/pix2pix-tensorflow/commits/master. [Accessed 25 March 2018].

[71] A. Gray, “Demystifying Data Input to TensorFlow for Deep Learning,” Github, 29 Nov 2016. [Online]. Available: https://agray3.github.io/2016/11/29/Demystifying-Data-Input-to-TensorFlow-for-Deep-Learning.html. [Accessed 25 May 2018].

[72] fabrizioM, “How do we use tf.image.encode_jpeg to write out an image in TensorFlow?,” StackOverflow, 29 Oct 2016. [Online]. Available: https://stackoverflow.com/questions/40320271/how-do-we-use-tf-image-encode-jpeg-to-write-out-an-image-in-tensorflow. [Accessed 25 March 2018].

[73] jkschin, “Saving image files in Tensorflow,” Github, 15 Jan 2016. [Online]. Available: https://stackoverflow.com/questions/34783030/saving-image-files-in-tensorflow. [Accessed 25 March 2018].

[74] T. Veldhuizen, “Measures of image quality,” 16 Jan 1998. [Online]. Available: http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/VELDHUIZEN/node18.html. [Accessed 28 March 2018].

[75] Z. Wang, E. P. Simoncelli and A. C. Bovik, “Multiscale structural similarity for image quality assessment,” in *Signals, Systems and Computers*, 2003.

[76] bsautermeister, “SSIM / MS-SSIM for TensorFlow,” StackOverflow, 20 Aug 2016. [Online]. Available: https://stackoverflow.com/questions/39051451/ssim-ms-ssim-for-tensorflow. [Accessed 25 March 2018].

[77] A. Mittal, R. Soundararajan and A. C. Bovik, “Making a "Completely Blind" Image Quality Analyzer,” *IEEE Signal Processing Letters,* vol. 20, no. 3, pp. 209-212, 2013.

[78] cs231n, “Learning,” Github, [Online]. Available: http://cs231n.github.io/neural-networks-3/#anneal. [Accessed 29 March 2018].

[79] K. Panetta, “Neural Networks and Modern BI Platforms Will Evolve Data and Analytics,” Gartner, 16 January 2017. [Online]. Available: https://www.gartner.com/smarterwithgartner/nueral-networks-and-modern-bi-platforms-will-evolve-data-and-analytics/. [Accessed 24 March 2018].

[80] A. S. Walia, “Types of Optimization Algorithms used in Neural Networks to Optimize Gradient Descent,” Towards Data Science, 10 Jun 2017. [Online]. Available: https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f. [Accessed 24 March 2018].

[81] S. Bianco, “Single and Multiple Illuminant Estimation Using Convolutional Neural Networks,” *IEEE Transactions on Image Processing,* vol. 26, no. 9, pp. 4347-4362, 2017.